

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-338684
 (43)Date of publication of application : 10.12.1999

(51)Int.Cl. G06F 9/06

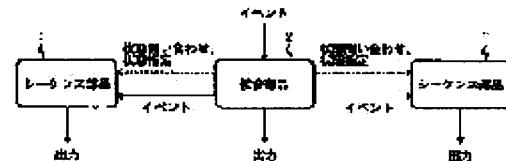
(21)Application number : 10-146247 (71)Applicant : TOSHIBA CORP
 (22)Date of filing : 27.05.1998 (72)Inventor : IKEDA NOBUYUKI

(54) STATE TRANSITION MOVEMENT SYSTEM, ITS GENERATING METHOD AND COMPUTER READABLE MEDIUM FOR RECORDING THE SAME SYSTEM

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a state transition movement system which efficiently develops a new state transition movement system by reusing an existing state transition movement system.

SOLUTION: A sequence control system is provided with at least two sequence parts 1 to realize partial movement specifications that are made by dividing a representable movement specification as a state transition diagram, and a connecting part 2 that connects the sequence parts 1. The part 2 switches activation of the parts 1 based on an external event. Further, the part 2 allocates the externally given event to either of each sequence part 1 when the event does not accompany activation switching between the parts 1 and switches activation of the parts 1 and also changes the internal state of the parts 1 when the event accompanies activation switching between the parts 1.



(19)日本国特許庁 (JP)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開平11-338684

(43)公開日 平成11年(1999)12月10日

(51)Int.Cl.⁸
G 0 6 F 9/06

識別記号
5 3 0

F I
G 0 6 F 9/06

5 3 0 T

審査請求 未請求 請求項の数6 O L (全25頁)

(21)出願番号 特願平10-146247

(22)出願日 平成10年(1998)5月27日

(71)出願人 000003078

株式会社東芝

神奈川県川崎市幸区堀川町72番地

(72)発明者 池田信之

東京都府中市東芝町1番地 株式会社東芝

府中工場内

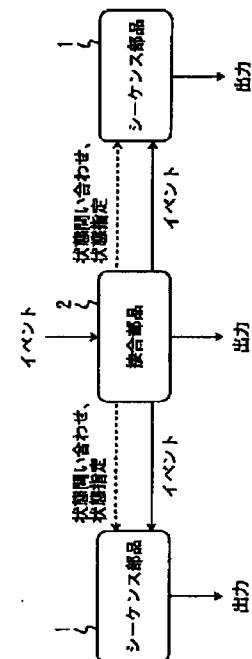
(74)代理人 弁理士 佐藤一雄 (外3名)

(54)【発明の名称】 状態遷移動作システム、その生成方法および状態遷移動作システムを記録したコンピュータ読み取り可能な記録媒体

(57)【要約】

【課題】 既存の状態遷移動作システムを再利用して新たな状態遷移動作システムを効率的に開発することを可能にする状態遷移動作システム等を提供する。

【解決手段】 シーケンス制御システムは、状態遷移図として表現可能な動作仕様を分割してなる部分的な動作仕様を実現する少なくとも2つのシーケンス部品1と、各シーケンス部品1間を接続する接合部品2とを備え、接合部品2は、外部から与えられたイベントに基づいてシーケンス部品1の起動を切り替えることができるようになっている。なお、接合部品2は、外部から与えられたイベントが各シーケンス部品1間の起動の切替えを伴わないときにはイベントを各シーケンス部品1のいずれかに振り分け、イベントが各シーケンス部品1間の起動の切替えを伴うときには各シーケンス部品1の起動を切り替えるとともに、各シーケンス部品1の内部状態を変更する。



【特許請求の範囲】

【請求項1】状態遷移図として表現可能な動作仕様を分割してなる部分的な動作仕様を実現する少なくとも2つの状態遷移部品と、

前記各状態遷移部品間を接続する接合部品とを備え、前記接合部品は、外部から与えられたイベントが前記各状態遷移部品間の起動の切替えを伴わないときには前記イベントを前記各状態遷移部品のいずれかに振り分け、前記イベントが前記各状態遷移部品間の起動の切替えを伴うときには前記各状態遷移部品の起動を切り替えるとともに、切替え先となる状態遷移部品の内部状態を変更することを特徴とする状態遷移動作システム。

【請求項2】前記各状態遷移部品は、外部から与えられたイベントを受け付けるイベント入力手段と、現在の内部状態を保持する現在状態ストア手段と、前記イベント入力手段により受け付けられたイベントに基づいて、前記現在状態ストア手段に保持されている内部状態を変更する状態変更手段と、外部から与えられた状態の指定に応じて、前記現在状態ストア手段に保持されている内部状態を変更する状態指定手段と、

外部から与えられた状態の問い合わせに応じて、前記前記現在状態ストア手段に保持されている現在の内部状態を返す状態問い合わせ手段とを有し、

前記接合部品は、外部から与えられたイベントを受け付けるイベント入力手段と、

アクティブとなっている状態遷移部品を表すデータを保持するアクティブ部品データストア手段と、前記イベント入力手段により受け付けられたイベントと、前記アクティブ部品データストア手段に保持されているデータに基づいて、イベントの振分け、前記各状態遷移部品の起動の切替え、および切替え先となる状態遷移部品の内部状態の変更を行うイベント対応処理手段と、

前記各状態遷移部品との間でイベント、内部状態の変更要求、および内部状態の問い合わせ要求の通信を行う部品通信手段とを有することを特徴とする請求項1記載の状態遷移動作システム。

【請求項3】前記各状態遷移部品は、前記イベント入力手段により受け付けられたイベントに基づいて、状態の遷移に伴うアクションを起動するアクション起動手段と、

前記アクション起動手段により起動されたアクションの動作内容を実現するアクション出力手段とをさらに有し、

前記接合部品は、前記イベント対応処理手段により前記各状態遷移部品の起動の切替えが行われるときに起動されるアクションの動作内容を実現するアクション出力手段をさらに有することを特徴とする請求項2記載の状態遷移動作システム。

【請求項4】前記各状態遷移部品は、前記現在状態ストア手段に保持されている内部状態を初期化する初期化手段をさらに有し、

前記接合部品は、前記アクティブ部品データストア手段に保持されている状態遷移部品を表すデータを初期化するとともに、前記部品通信手段により通信が行われる通信路を確保する初期化手段をさらに有することを特徴とする請求項2または3記載の状態遷移動作システム。

【請求項5】第1の状態遷移部品の動作仕様に基づいて状態の遷移に対応するイベント、遷移元の状態、遷移先の状態およびアクションを含む遷移データを抽出して第1の遷移データリストを作成するステップと、要求される状態遷移動作システムの動作仕様に基づいて状態の遷移に対応するイベント、遷移元の状態、遷移先の状態およびアクションを含む遷移データを抽出して第2の遷移データリストを作成するステップと、

前記第1の遷移データリストと前記第2の遷移データリストとの間に共通の遷移データが存在するときに、前記第2の遷移データリストに含まれる遷移データの中から前記第1の遷移データリストに含まれる遷移元または遷移先の状態を含まない遷移データを抽出して第3の遷移データリストを作成するステップと、

前記第2の遷移データリストから前記第1の遷移データリストおよび前記第3の遷移データリストと重複している部分を取り除いて第4の遷移データリストを作成するステップと、

前記第3の遷移データリストに基づいて前記第1の状態遷移部品とともに用いられる第2の状態遷移部品を作成するステップと、

30 前記第4の遷移データリストに基づいて前記第1の状態遷移部品および前記第2の状態遷移部品間を接続する接合部品を作成するステップとを含むことを特徴とする状態遷移動作システムの生成方法。

【請求項6】状態遷移図として表現可能な動作仕様を分割してなる部分的な動作仕様を実現する少なくとも2つの状態遷移部品と、

前記各状態遷移部品間を接続する接合部品とを備え、前記接合部品は、外部から与えられたイベントが前記各状態遷移部品間の起動の切替えを伴わないときには前記イベントを前記各状態遷移部品のいずれかに振り分け、前記イベントが前記各状態遷移部品間の起動の切替えを伴うときには前記各状態遷移部品の起動を切り替えるとともに、切替え先となる状態遷移部品の内部状態を変更することを特徴とする状態遷移動作システムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は状態遷移図として表現可能な動作仕様を実現するシーケンス制御システム等の状態遷移動作システムに係り、とりわけ既存の状態遷

移動作システムを再利用して新たな状態遷移動作システムを効率的に開発することを可能にする状態遷移動作システム、その生成方法および状態遷移動作システムを記録したコンピュータ読み取り可能な記録媒体に関する。

【0002】

【従来の技術】従来から、TV制御システム等のように外部からの入力や内部状態の変化に起因した一連の動作を制御するシステムとしてシーケンス制御システムが知られている。このようなシーケンス制御システムの動作仕様は通常、状態遷移図により記述することができる。状態遷移図は、状態間の遷移、遷移を引き起こす入力（イベント）や遷移を引き起こす条件、および遷移に伴う動作（アクション）等を図式的に表現したものである。なお、状態遷移図により記述される動作仕様を実現するシーケンス制御システムはコンピュータプログラムにより実装することができる。

【0003】ところで、コンピュータプログラムによりシーケンス制御システムを実装する場合には、過去に開発したシーケンス制御システムの動作仕様に新たに機能を追加してその動作仕様を拡張するという要求がしばしば発生する。従来においては、このような場合、状態間の遷移等を実現するシーケンス制御プログラムの内部を直接変更することにより、新たなシーケンス制御プログラムを作成していた。

【0004】図23乃至図26は従来のシーケンス制御システム（TV制御システム）の開発方法を説明するための図である。図23はTV制御システムの動作仕様を記述した状態遷移図である。図23において、角の丸い2つのボックスは、TV制御システムがとりうる“電源OFF”と“選局処理”という2つの状態を表している。また、ボックスをつなぐ矢印は状態間の遷移を表し、矢印に添えられた記述はその遷移を発生させるイベントとその遷移に伴って実行されるアクションとを表している。なおここでは、3つの遷移が存在している。

“電源OFF”状態から“選局処理”状態への遷移は“電源ボタン入力”イベントを受け付けたときに発生し、同時にアクションとして“電源リレーオン”の動作が行われる。また、“選局処理”状態から“選局処理”状態への遷移は“選局ボタン入力”イベントを受け付けたときに発生し、同時にアクションとして“チャンネル切替え処理”の動作が行われる。さらに、“選局処理”状態から“電源OFF”状態への遷移は“電源ボタン入力”イベントを受け付けたときに発生し、同時にアクションとして“電源リレーオフ”的動作が行われる。

【0005】すなわち、図23に示す状態遷移図には、電源ボタンと選曲ボタンとを有する単純なTVの動作仕様が記述されており、電源ボタンの入力によって電源が入り切りされるとともに、電源ON時には選曲ボタンによってチャンネルの切替えが可能であることが示されている。なお、このような動作仕様を実現するシーケンス

制御システムは例えば、プログラミング言語（C++言語）により図24に示すようなシーケンス制御プログラムとして実装される。

【0006】ここで、このようなシーケンス制御システムに新たに機能を追加してその動作仕様を拡張するという要求が発生した場合を考える。図25は拡張された動作仕様の一例を記述した状態遷移図である。図25はお休みタイマボタンを押してから一定時間後に電源をオフする“お休みタイマ”的機能が追加された場合を示しており、この場合には図23に示す状態遷移図に対して新たに4つの遷移と2つの状態とが追加されている。ここで、“選局処理”状態から“お休みタイマ待ち”状態への遷移は“選局処理”状態で“お休みタイマボタン入力”イベントを受け付けたときにのみ発生し、同時にアクションとして“お休みタイマ起動”的動作が行われる。また、“お休みタイマ待ち”状態と“時間表示”状態との間の双方向の遷移は“お休みタイマ待ち”状態で“表示ボタン入力”イベントを受け付けたときに発生する。“お休みタイマ待ち”状態で“表示ボタン入力”イベントを受け付けると、アクションとして“電源OFFまでの時間表示”および“表示タイマ起動”的動作が行われ、表示タイマのカウントが終了するまでの一定時間の間、画面上に電源OFFまでの時間が表示される。さらに、“お休みタイマ待ち”状態から“電源OFF”状態への遷移はお休みタイマのカウントが終了した時点で発生し、アクションとして“電源リレーオフ”的動作が行われる。

【0007】なお、従来においては、このような拡張された動作仕様を実現するシーケンス制御プログラムを開発するための方法として、図24に示すようなシーケンス制御プログラムの内部を直接変更することにより、図25に示すような動作仕様を実現する新たなシーケンス制御プログラムを作成していた。図26はこのような方法により作成されたシーケンス制御プログラムの一例を示す図である。なお、図26に示すシーケンス制御プログラムにおいては、3～4行、11～12行および34～71行が変更されている。

【0008】

【発明が解決しようとする課題】上述したように、従来においては、過去に開発したシーケンス制御システムの動作仕様に新たに機能を追加してその動作仕様を拡張する場合に、既存のシーケンス制御プログラムの内部を直接変更することにより、新たなシーケンス制御プログラムを作成していた。

【0009】しかしながら、このような従来のシーケンス制御システムの生成方法では、既存のシーケンス制御プログラムの内部に直接手を入れて修正することとなるので、修正対象となるシーケンス制御プログラムを詳細に理解する必要がある。このため、修正対象となるシーケンス制御プログラムの内部が複雑である場合には、多

くの作業工数が必要となり、また作業中に誤りが混入する可能性が増加するという問題がある。

【0010】また、シーケンス制御プログラムがマスクROM等を利用してハードウェア上で実現されているような場合には、シーケンス制御プログラムを再利用する場合であっても既存のROMを廃棄して新たにROMを作成しなければならず、経済的でないという問題がある。

【0011】本発明はこのような点を考慮してなされたものであり、既存の状態遷移動作システムを再利用して新たな状態遷移動作システムを効率的に開発することを可能にする状態遷移動作システム、その生成方法および状態遷移動作システムを記録したコンピュータ読み取り可能な記録媒体を提供することを目的とする。

【0012】

【課題を解決するための手段】本発明の第1の特徴は、状態遷移図として表現可能な動作仕様を分割してなる部分的な動作仕様を実現する少なくとも2つの状態遷移部品と、前記各状態遷移部品間を接続する接合部品とを備え、前記接合部品は、外部から与えられたイベントが前記各状態遷移部品間の起動の切替えを伴わないときには前記イベントを前記各状態遷移部品のいずれかに振り分け、前記イベントが前記各状態遷移部品間の起動の切替えを伴うときには前記各状態遷移部品の起動を切り替えるとともに、切替え先となる状態遷移部品の内部状態を変更することを特徴とする状態遷移動作システムである。

【0013】本発明の第2の特徴は、第1の状態遷移部品の動作仕様に基づいて状態の遷移に対応するイベント、遷移元の状態、遷移先の状態およびアクションを含む遷移データを抽出して第1の遷移データリストを作成するステップと、要求される状態遷移動作システムの動作仕様に基づいて状態の遷移に対応するイベント、遷移元の状態、遷移先の状態およびアクションを含む遷移データを抽出して第2の遷移データリストを作成するステップと、前記第1の遷移データリストと前記第2の遷移データリストとの間に共通の遷移データが存在するときに、前記第2の遷移データリストに含まれる遷移データの中から前記第1の遷移データリストに含まれる遷移元または遷移先の状態を含まない遷移データを抽出して第3の遷移データリストを作成するステップと、前記第2の遷移データリストから前記第1の遷移データリストおよび前記第3の遷移データリストと重複している部分を取り除いて第4の遷移データリストを作成するステップと、前記第3の遷移データリストに基づいて前記第1の状態遷移部品とともに用いられる第2の状態遷移部品を作成するステップと、前記第4の遷移データリストに基づいて前記第1の状態遷移部品および前記第2の状態遷移部品間を接続する接合部品を作成するステップとを含むことを特徴とする状態遷移動作システムの生成方法で

ある。

【0014】本発明の第3の特徴は、状態遷移図として表現可能な動作仕様を分割してなる部分的な動作仕様を実現する少なくとも2つの状態遷移部品と、前記各状態遷移部品間を接続する接合部品とを備え、前記接合部品は、外部から与えられたイベントが前記各状態遷移部品間の起動の切替えを伴わないときには前記イベントを前記各状態遷移部品のいずれかに振り分け、前記イベントが前記各状態遷移部品間の起動の切替えを伴うときには前記各状態遷移部品の起動を切り替えるとともに、切替え先となる状態遷移部品の内部状態を変更することを特徴とする状態遷移動作システムを記録したコンピュータ読み取り可能な記録媒体である。

【0015】本発明の第1乃至第3の特徴によれば、状態遷移動作システムを再利用して新たな状態遷移動作システムを生成することができるので、新たな状態遷移動作システムを効率的に開発することができる。また、このような開発過程で作成された状態遷移部品や接合部品についても次の新たな開発過程でシーケンス部品として再利用することができるので、状態遷移動作システムをサブシステム化（部品化）してシステム開発を効率的に進めることができる。

【0016】

【発明の実施の形態】以下、図面を参照して本発明の実施の形態について説明する。図1乃至図12は本発明による状態遷移動作システムおよびその生成方法の一実施の形態を示す図である。なお、本実施の形態においては、状態遷移動作システムをシーケンス制御システムに適用した場合を例にとり説明する。

【0017】まず、図1乃至図3により、シーケンス制御システムの構成について説明する。

【0018】図1に示すように、シーケンス制御システムは、状態遷移図として表現可能な動作仕様を分割してなる部分的な動作仕様を実現する少なくとも2つのシーケンス部品（状態遷移部品）1と、各シーケンス部品1間を接続する接合部品2とを備え、接合部品2は、外部から与えられたイベントに基づいてシーケンス部品1の起動を切り替えることができるようになっている。

【0019】このうち、各シーケンス部品1は、図2に示すように、外部から与えられたイベントを受け付けるイベント入力手段1aと、現在の内部状態を保持する現在状態ストア手段1eと、イベント入力手段1aにより受け付けられたイベントに基づいて、現在状態ストア手段1eに保持されている内部状態を変更するとともに状態の遷移に伴うアクションを起動する状態変更／アクション起動手段1dと、状態変更／アクション起動手段1dにより起動されたアクションの動作内容を実現するアクション出力手段1fとを有している。また、各シーケンス部品1は、外部から与えられた状態の指定に応じて、現在状態ストア手段1eに保持されている内部状態

を変更する状態指定手段1 bと、外部から与えられた状態の問い合わせに応じて、現在状態ストア手段1 eに保持されている現在の内部状態を返す状態問い合わせ手段1 cと、現在状態ストア手段1 eに保持される内部状態を初期化する初期化手段1 gと有している。

【0020】一方、接合部品2は、図3に示すように、外部から与えられたイベントを受け付けるイベント入力手段2 aと、アクティブとなっているシーケンス部品を表すデータを保持するアクティブ部品データストア手段2 cと、イベント入力手段2 aにより受け付けられたイベントと、アクティブ部品データストア手段2 cに保持されているデータとに基づいて、イベントの振分け、各シーケンス部品1の起動の切替え、および切替え先となるシーケンス部品1の内部状態の変更を行うイベント対応処理手段2 bとを有している。また、接合部品2は、各シーケンス部品1との間でイベント、内部状態の変更要求、および内部状態の問い合わせ要求の通信を行う部品通信手段2 dを有している。さらに、接合部品2は、イベント対応処理手段2 bにより各シーケンス部品1の起動の切替えが行われるときに必要に応じて起動されるアクションの動作内容を実現するアクション出力手段2 eと、アクティブ部品データストア手段1 cに保持されているシーケンス部品を表すデータを初期化して部品利用開始時にアクティブとなるシーケンス部品を設定するとともに、部品通信手段2 dにより通信が行われる通信路を確保する初期化手段2 fとを有している。

【0021】次に、図4乃至図8により、このような構成からなるシーケンス制御システムの生成方法について説明する。なお、ここでは、図4に示すような動作仕様を実現する既存のシーケンス制御システムをシーケンス部品として再利用して、プログラミング言語(C++言語)により、図5に示すような動作仕様を実現するシーケンス制御システムを生成する場合を例にとり説明する。

【0022】図6に示すように、まず、既存のシーケンス部品(第1の状態遷移部品)の動作仕様に基づいて状態の遷移に対応するイベント、遷移元の状態、遷移先の状態およびアクションを含む遷移データを抽出して部品シーケンス仕様(第1の遷移データリスト)を作成する(ステップ101)。

【0023】ここで、「シーケンス仕様」とは、動作仕様を記述した状態遷移図と同等の内容を集合の概念を用いて表現したものである。具体的には、状態遷移図における状態の遷移を、イベント、遷移元の状態、遷移先の状態およびアクションの4つの要素を含む遷移データで表現し、動作仕様全体はこの遷移データの集合として表現する。ここで、図4に示すような状態遷移図をシーケンス仕様で表現すると次の仕様1のようになる。

【0024】仕様1： {<event1, α , β , action1>, <event2, β , α , action2>}

なお、図4に示すような状態遷移図により記述される動作仕様は、再利用しようとする既存のシーケンス制御システム(シーケンス部品)の動作仕様であるので、ここでは上記仕様1を部品シーケンス仕様と呼ぶ。

【0025】次に、要求されるシーケンス制御システムの動作仕様に基づいて状態の遷移に対応するイベント、遷移元の状態、遷移先の状態およびアクションを含む遷移データを抽出して要求シーケンス仕様(第2の遷移データリスト)を作成する(ステップ102)。

10 【0026】ここで、要求シーケンス仕様とは、これから実現しようとするシーケンス制御システムの動作仕様をシーケンス仕様として表現したものである。ここで、図5に示す状態遷移図により記述される動作仕様を実現しようとする場合には、要求シーケンス仕様は次の仕様2のようになる。

仕様2： {<event1, α , β , action1>, <event2, β , α , action2>, <event3, γ , α , action3>, <event4, β , γ , action4>, <event1, γ , δ , action5>, <event2, δ , γ , action6>}

20 【0027】次に、既存のシーケンス部品が再利用可能であるか否かを判定する(ステップ103)。ここで、要求されるシーケンス制御システムを実現するときに既存のシーケンス部品が再利用可能であるためには、部品シーケンス仕様に含まれる遷移データと要求シーケンス仕様に含まれる遷移データとの間に共通の遷移データが存在する必要がある(再利用条件)。なお、共通の遷移データとは、遷移元の状態、遷移先の状態およびアクションの4つの要素が全て等しいという意味である。

【0028】上述したステップ103において、部品シーケンス仕様と要求シーケンス仕様との間に共通の遷移データが存在して既存のシーケンス部品が再利用可能であることが判定されたときには、要求シーケンス仕様に含まれる遷移データの中から部品シーケンス仕様に現れる遷移元または遷移先の状態を含まない遷移データを抽出して差分シーケンス仕様(第3の遷移データリスト)を作成する(ステップ104)。

【0029】ここで、差分シーケンス仕様とは、要求シーケンス仕様と部品シーケンス仕様との差分を抽出したシーケンス仕様である。具体的には、要求シーケンス仕

40 様に含まれる遷移データの中から部品シーケンス仕様に現れる遷移元または遷移先の状態を含まない遷移データを抽出したものである。なお、上記仕様1を部品シーケンス仕様、上記仕様2を要求シーケンス仕様とする場合には、差分シーケンス仕様は次の仕様3のようになる(図7参照)。

仕様3： {<event1, γ , δ , action5>, <event2, δ , γ , action6>}

次に、要求シーケンス仕様から部品シーケンス仕様および差分シーケンス仕様と重複している部分を取り除いて接合シーケンス仕様(第4の遷移データリスト)を作成

する（ステップ105）。

【0030】ここで、接合シーケンス仕様とは、部品シーケンス仕様を実現するシーケンス部品と差分シーケンス仕様を実現するシーケンス部品とを接続するためのシーケンス仕様である。具体的には、要求シーケンス仕様のうち部品シーケンス仕様にも差分シーケンス仕様にも含まれない遷移データの集合である。なお、なお、上記仕様1を部品シーケンス仕様、上記仕様2を要求シーケンス仕様とする場合には、接合シーケンスは次の仕様4のようになる（図8参照）。

仕様4： {<event3,γ,α,action3>, <event4,β,γ,action4>}

【0031】その後、差分シーケンス仕様に基づいて既存のシーケンス部品とともに用いられる新たなシーケンス部品（第2の状態遷移部品）を作成するとともに（ステップ106）、接合シーケンス仕様に基づいて既存のシーケンス部品および新たなシーケンス部品間を接続する接合部品を作成する（ステップ107）。

【0032】ここで、部品シーケンス仕様に対応する既存のシーケンス部品、差分シーケンス仕様に対応する新たなシーケンス部品、および接合シーケンス仕様に対応する接合部品は、例えばプログラミング言語（C++言語）によりプログラムとして実装される。なお、これらのプログラムは、コンピュータ上のメモリやハードディスク等の内部記憶装置、およびフレキシブルディスクやCD-ROM等の外部記憶装置のようなコンピュータ読み取り可能な各種の記録媒体に格納され、コンピュータ上のCPU（中央演算処理装置）から逐次読み出されて実行される。

【0033】部品シーケンス仕様または差分シーケンス仕様のいずれかに対応するシーケンス部品1は、図2に示すように、イベント入力手段1a、状態指定手段1b、状態問い合わせ手段1c、状態変更／アクション起動手段1d、現在状態ストア手段1e、アクション出力手段1f、および初期化手段1gを有している。

【0034】図2に示すシーケンス部品1において、イベント入力手段1aは外部からシーケンス部品1に対してイベントを入力するためのものであり、プログラム上ではイベントの発生時に外部から起動される関数や、OSやプログラミング言語の提供するイベントディスパッチャ関数等により実現される。イベント入力手段1aはイベントの入力を状態変更／アクション起動手段1dに通知する。状態変更／アクション起動手段1dは入力されたイベントに基づいて内部状態の変更とアクションの起動とを部品シーケンス仕様または差分シーケンス仕様に従って行うためのものであり、現在状態ストア手段1eに保持されている現在の内部状態と、入力されたイベントに基づいて、遷移すべき次の状態と起動すべきアクションとを求め、状態の変更が必要な場合には現在状態ストア手段1eに保持されている内部状態を次の状態に

書き換え、またアクションの起動が必要な場合にはアクション出力手段1fにアクションの起動を通知する。アクション出力手段1fは個々のアクションの動作内容を実現するためのものであり、プログラム上ではアクションの動作内容を実現する関数等で実現され、状態変更／アクション起動手段1dによるアクションの起動の通知はこれらの関数の呼出し等により実現される。現在状態ストア手段1eはプログラム上では現在の内部状態を保持する変数として実現される。状態指定手段1bは、外部から状態が指定された場合に現在状態ストア手段1eに保持されている値を指定状態に対応する値に書き換えるためのものであり、プログラム上では変数である現在状態ストア手段1eに保持されている値を書き換える関数等により実現される。状態問い合わせ手段1cは現在の内部状態を外部に通知するためのものであり、プログラム上では変数である現在状態ストア手段1eに保持されている値を参照して現在の内部状態を表す値を返す関数や、引数として与えられた状態に対応する値に対して現在がその状態であるか否かをブール値で返す関数等により実現される。

【0035】一方、接合シーケンス仕様に対応する接合部品2は、図3に示すように、イベント入力手段2a、イベント対応処理手段2b、アクティブ部品データストア手段2c、部品通信手段2d、アクション出力手段2e、および初期化手段2fを有している。

【0036】図3に示す接合部品2において、イベント入力手段2aは外部から接合部品2に対してイベントを入力するためのものであり、プログラム上ではイベント発生時に外部から起動される関数や、OSやプログラミング言語の提供するイベントディスパッチャ関数等により実現される。イベント入力手段2aは要求シーケンス仕様に含まれる全てのイベントを受け付けるものであり、例えば1つのイベントの入力を1つの関数の呼出しによって行うような場合には要求シーケンス仕様に含まれる全てのイベントに対応する関数を作成する。イベント入力手段2aはイベントの入力をイベント対応処理手段2bに通知する。イベント対応処理手段2bはイベント入力手段2aにより受け付けられたイベントが各シーケンス部品1間の起動の切替えを伴わないときにはイベントを各シーケンス部品1のいずれかに振り分け、イベントが各シーケンス部品1間の起動の切替えを伴うときには各シーケンス部品1の起動を切り替えるとともに、各シーケンス部品1の内部状態を変更する。

【0037】図9はイベント入力手段2aおよびイベント対応処理手段2b等における処理内容を説明するためのフローチャートである。図9に示すように、初期化手

段2 f による初期化処理が終了した後（ステップ2 0 1）、イベント入力手段2 a によりイベントを受け付ける（ステップ2 0 2）。そして、イベント対応処理手段2 b において、入力されたイベントと、アクティブ部品データストア手段2 e に保持されている現在アクティブとなっているシーケンス部品を表すデータとに基づいて部品切替え条件を判定する（ステップ2 0 3）。ここで、部品切替え条件とは、分割された部品シーケンス仕様および差分シーケンス仕様のそれぞれを実現する各シーケンス部品を切り替えて動作させる必要があるか否かを判定するものである。具体的には、入力されたイベントをe、アクティブなシーケンス部品の現在の状態をS とするとき、e をイベント（第1要素）、S を遷移元の状態（第2要素）とする遷移データが接合シーケンス仕様に存在するか否かを判定する。

【0038】ここで、部品切替え条件が成立した場合には、該当する遷移データのアクションをアクション出力手段2 e により実行するとともに（ステップ2 0 4）、切替え先となるシーケンス部品1 の内部状態を、部品通信手段2 d およびシーケンス部品1 の状態指定手段1 b により、該当する遷移データの遷移先の状態（第3要素）に変化させる（ステップ2 0 5）。また同時に、切替え先となるシーケンス部品を表すデータをアクティブ部品データストア手段2 c に設定する（ステップ2 0 6）。なお、ステップ2 0 4 乃至2 0 6 の処理の順序は任意である。

【0039】一方、部品切替え条件が成立しない場合には、アクティブ部品データストア手段2 c を参照してアクティブなシーケンス部品を求め、部品通信手段2 d により、そのアクティブなシーケンス部品に対してイベントを送信する。

【0040】なお、初期化手段2 f は、プログラム上ではアクティブ部品データストア手段2 c に保持されているデータを部品利用開始時にアクティブとなるシーケンス部品を表すデータに書き換えるとともに、部品通信手段2 d により通信が行われる通信路を確保する関数等により実現される。

【0041】図10乃至図12は部品シーケンス仕様に対応する既存のシーケンス部品、差分シーケンス仕様に対応する新たなシーケンス部品、および接合シーケンス仕様に対応する接合部品のそれぞれをプログラミング言語（C++言語）によりプログラムとして実装した場合の一例を示す図である。

【0042】図10に示すクラスComponentは、上記仕様1の部品シーケンス仕様（図4参照）を実現するシーケンス部品1を実装するためのプログラムである。図10において、イベント入力手段1 a は関数event1(), event2()に対応している。外部からのイベントの入力はこれらの関数を呼び出すことにより行われる。アクション出力手段1 f は関数action1(), action2()に対応してい

る。これらの関数の内部では、例えば出力バッファへのデータの出力やタイマの起動等の様々なアクションが実装されるが、ここでは詳細な説明は省略する。現在状態ストア手段1 e は変数stateにより実現される。変数stateは値として“Alpha”または“Beta”をとり、それぞれの値をとったときにシーケンス部品Componentの現在の状態が“ α ”または“ β ”となる（図4参照）。初期化手段1 g は初期化関数Component()として実現され、これによりクラスComponentのインスタンスの生成時に変数stateの値が“Alpha”に設定される。このような初期化関数Component()の働きにより、シーケンス部品Componentの初期状態が“ α ”として設定される。状態変更/アクション出力手段1 d は関数event1(), event2()の関数本体プログラムにより実現されている。関数event1(), event2()は関数呼出しが発生する度に、現在のstateの値に基づいて次のstateへの変化を求め、状態の遷移が必要な場合にはその値を書き換える。状態指定手段1 b は関数setState()により実現される。この関数は引数として与えられた値をstateに代入することにより指定状態への変更を実現している。また、状態問い合わせ手段1 c は、関数isState()により実現される。この関数は現在の状態が引数として与えられた状態であるか否かを判定し、その結果を返す。

【0043】図11に示すクラスComplementは、上記仕様3の差分シーケンス仕様（図7参照）を実現するシーケンス部品1を実装するためのプログラムである。図11に示すように、このプログラムの構造は図10に示すプログラムの構造と略同一であるので、詳細な説明は省略する。

【0044】図12に示すクラスSwitcherは、上記仕様4の接合シーケンス仕様（図8参照）を実現する接合部品2を実装するためのプログラムである。図12において、イベント入力手段2 a は関数event1()～event4()に対応している。外部からのイベントの入力はこれらの関数を呼び出すことにより行われる。アクティブ部品データストア手段2 c は変数ComponentActiveFlag, ComplementActiveFlagにより実現される。これらの変数はそれぞれシーケンス部品Component, Complementに対応し、値が1であるときにはそのシーケンス部品がアクティブであることを表し、値が0であるときにはそのシーケンス部品がアクティブでないことを表している。部品通信手段2 d はシーケンス部品へのポインタ変数であるcomponent, complementにより実現される。これらにはそれぞれシーケンス部品Componentのインスタンスと、シーケンス部品Complementのインスタンスのアドレスとが保持される。C++言語ではポインタ変数を介してインスタンスの関数を呼び出すことができる。このため、接合部品Switcherはそれぞれのインスタンスの関数として実現されたシーケンス部品のイベント入力手段1 a 、状態指定手段1 b および状態問い合わせ手段1 c にこれらの

ポインタ変数を用いてアクセスすることができる。

【0045】初期化手段2 f は初期化関数Switcher()として実現される。初期化関数Switcher()は引数としてシーケンス部品Componentのインスタンスのアドレスと、シーケンス部品Complementのインスタンスのアドレスとをとり、それぞれを部品通信手段2 d であるポインタ変数component, complementに代入する。これにより、部品通信手段2 d は初期化され、シーケンス部品Component, Complementにアクセスするために利用することができる。また、初期化関数Switcher()はアクティブ部品データストア手段2 c である変数ComponentActiveFlag, ComplementActiveFlagの値をそれぞれ1 および0 に設定する。これにより、アクティブなシーケンス部品をComponentに初期化する。

【0046】イベント対応処理手段2 b は関数event3(), event4()の関数本体プログラムにより実現される。部品切替え条件の判定は、関数event3(), event4()の内部の2つのif文で実現される。部品切替え条件の成立時のアクションの実行の処理、切替え先となるシーケンス部品の内部状態を変更する処理、および切替え先となるシーケンス部品をアクティブとして他をインアクティブとする処理は、関数event3()の関数本体プログラムでは関数action3()の呼び出し、ポインタ変数componentで指定されるシーケンス部品への関数setState()の呼び出し、変数ComponentActiveFlag, ComplementActiveFlagの書き換えにより実現される。また、関数event4()の関数本体プログラムでは関数action4()の呼び出し、ポインタ変数complementで指定されるシーケンス部品への関数setState()の呼び出し、変数ComponentActiveFlag, ComplementActiveFlagの書き換えにより実現される。一方、部品切替え条件の不成立時の処理は、関数event1(), event2()の内部におけるアクティブなシーケンス部品へのイベント受付け関数event1(), event2()の呼び出しにより実現される。なお、アクション出力手段2 e には、関数action3(), action4()が対応している。

【0047】このように本実施の形態によれば、図4に示すような動作仕様（部品シーケンス仕様）を実現する既存のシーケンス制御システムを再利用して、図5に示すような動作仕様（要求シーケンス仕様）を実現するシーケンス制御システムを生成することができるので、新たなシーケンス制御システムを効率的に開発することができる。

【0048】また、このような開発過程で作成されたシーケンス部品（図11参照）や接合部品（図12参照）についても次の新たな開発過程でシーケンス部品として再利用することができるので、シーケンス制御システムをサブシステム化（部品化）してシステム開発を効率的に進めることができる。

【0049】なお、上述した実施の形態においては、本発明をシーケンス制御システムに適用した場合について

説明したが、これに限らず、コンパイラの字句解析システムのような各種の状態遷移動作システムについても同様にして適用することができる。

【0050】

【実施例】次に、図1乃至図12に示すシーケンス制御システムの具体的実施例について述べる。

【0051】第1の実施例

まず、図13乃至図19により、図1乃至図12に示すシーケンス制御システムをTV制御システムに適用した実施例について説明する。なお、ここでは、図13に示すような状態遷移図により記述される動作仕様を実現するシーケンス制御システムを開発した後、このシーケンス制御システムをシーケンス部品として再利用して、図14に示すような状態遷移図により記述される動作仕様を実現するシーケンス制御システムを生成する場合について説明する。

【0052】図15は図13に示す動作仕様を実現するシーケンス部品TvSequence1をプログラミング言語（C++言語）により実装した場合の一例を示す図である。

図13に示す状態遷移図においては、“電源ボタン入力”と“選局ボタン入力”という2種類のイベントが存在するので、これらのイベントを受け付けるようイベント入力手段1 aを作成する。イベントの入力はプログラム上では関数の呼び出しに対応させることができる。図15に示すプログラムにおいては、イベント入力手段1 aとして2つの関数powerButtonEvent(), selectButtonEvent()を有し、それぞれ外部からの“電源ボタン入力”イベントおよび“選局ボタン入力”イベントの入力をこれらの関数の外部からの呼び出しとして受け付けることができる。イベントが入力されると、イベントに対応した状態の変更とアクションの起動とが状態変更／アクション起動手段1 dにより行われるが、図15に示すプログラムでは関数powerButtonEvent(), selectButtonEvent()の関数本体プログラムがこれに対応する。例えば、関数powerButtonEvent()の関数本体プログラムとして実現されている状態変更／アクション起動手段1 dでは、if文による条件分岐により、変数stateが“POWER_OF”であるか“TUNING”であるかに基づいて、現在の状態が“電源オフ”状態であるか“選局処理”状態であるかが判定され、それぞれの状態に応じてイベントに対応する状態の変更とアクションの起動とが行われるようになっている。

【0053】また、アクションの動作内容を実現するアクション出力手段1 fは、“電源リレーオン”、“電源リレーオフ”および“チャンネル切替え処理”的アクションにそれぞれ対応する関数powerOn(), powerOff(), setChannel(char ch)として実現されている。それぞれの関数の関数本体プログラムはここでは省略しているが、TV制御システムでは実デバイスである制御用ICに対して動作を指令するデータを送信するためのプログラム

やそのようなサブルーチンの呼出しといった実際の処理が実装される。シーケンス部品の現在の状態を保持する現在状態ストア手段 1 e は変数 state として実現されている。変数 state は列挙型 State の変数であり、値として "POWER_OFF" または "TUNING" をとるが、それぞれの値をとったときにシーケンス部品の現在の状態が "電源 OFF" 状態または "選局処理" 状態となることが示される。外部から状態を強制的に変更する状態指定手段 1 b と、外部から状態について問い合わせを行う状態問い合わせ手段 1 c はそれぞれ関数 setState(State tmpState), isState(State tmpState) として実現されている。関数 setState(State tmpState) は引数として与えられた値を関数本体プログラム中で変数 state に代入することにより指定状態への変更を実現している。また、関数 isState(State tmpState) は引数として与えられた値を関数本体プログラム中で現在の状態の値を保持する変数 state と比較し、その結果を真理値として返すことにより、外部の関数の呼出し側に現在の状態についての情報を受け渡すようになっている。

【0054】なお、部品の状態を初期化する初期化手段 1 g は初期化関数 TvSequence1() として実現されており、ここでは関数本体プログラム中で変数 state に "POWER_OFF" の値を代入することにより、状態を "電源 OFF" 状態に初期化している。

【0055】次に、図 15 に示すような TV 制御システムが既に存在するときに、この TV 制御システムをシーケンス部品として再利用して、図 14 に示す状態遷移図により記述される動作仕様を実現する TV 制御システムを生成する手順について説明する。

【0056】図 6 に示すように、まず、既存のシーケンス部品の動作仕様に基づいて部品シーケンス仕様を作成する（ステップ 101）。具体的には、図 13 に示す状態遷移図中の遷移を抜き出し、個々の遷移ごとに遷移に対応するイベント、遷移元の状態、遷移先の状態およびアクションの 4 つの要素を含む遷移データとし、この遷移データの集合として部品シーケンス仕様を作成する。なお、図 13 に示す状態遷移図には "電源 OFF" 状態から "選局処理" 状態への遷移、"選局処理" 状態から "電源 OFF" 状態への遷移、"選局処理" 状態から "選局処理" 状態への遷移という 3 つの遷移が存在するので、部品シーケンス仕様はこれらの遷移に対応する 3 つの遷移データの集合として次の仕様 5 のように作成される。

仕様 5：{<電源ボタン入力, 電源 OFF, 選局処理, 電源リレーオン>, <電源ボタン入力, 選局処理, 電源 OFF, 電源リレーオフ>, <選局ボタン入力, 選局処理, 選局処理, チャンネル切替え処理>}

【0057】次に、要求される TV 制御システムの動作仕様に基づいて要求シーケンス仕様を作成する（ステップ 102）。具体的には、図 14 に示す状態遷移図に対

応する要求シーケンス仕様を作成する。作成の方法は上述したステップ 101 と同様であり、得られる要求シーケンス仕様は次の仕様 6 のようになる。この仕様 6 の要求シーケンス仕様には図 14 に示す状態遷移図に含まれる 7 つの遷移に対応して 7 つの遷移データがある。なお、6 番目の遷移データでは、2 つのアクションを括弧で括っているが、これは 1 つの遷移に対して 2 つのアクションがあることを表している。

仕様 6：{<電源ボタン入力, 電源 OFF, 選局処理, 電源リレーオン>, <電源ボタン入力, 選局処理, 電源 OFF, 電源リレーオフ>, <選局ボタン入力, 選局処理, 選局処理, チャンネル切替え処理><お休みタイマボタン入力, 選局処理, お休みタイマ待ち, お休みタイマ起動>, <お休みタイマカウント終了, お休みタイマ待ち, 電源 OFF, 電源リレーオン>, <表示ボタン入力, お休みタイマ待ち, 時間表示, (電源 OFFまでの時間表示, 表示タイマ起動)>, <表示タイマカウント終了, 時間表示, お休みタイマ待ち, 表示消去>}

【0058】次に、既存のシーケンス部品が再利用可能であるか否かを判定する（ステップ 103）。ここで、再利用条件とは、上述したように、部品シーケンス仕様に含まれる遷移データと要求シーケンス仕様に含まれる遷移データとの間に共通の遷移データが存在することである。ここで、部品シーケンス仕様である上記仕様 5 と要求シーケンス仕様である上記仕様 6 を比較すると、上記仕様 5 に含まれる 3 つの遷移データが上記仕様 6 の 3 番目までのそれぞれの遷移データと等しいことが分かる。従って、図 15 に示すシーケンス部品は再利用可能であると判定される。

【0059】この場合、上記仕様 6 の要求シーケンス仕様に含まれる遷移データの中から上記仕様 5 の部品シーケンス仕様に現れる遷移元または遷移先の状態を含まない遷移データを抽出して差分シーケンス仕様を作成する（ステップ 104）。ここで、上記仕様 5 の部品シーケンス仕様に現れる遷移元または遷移先の状態は "電源 OFF" 状態と "選局処理" 状態であり、これらの状態を含まない遷移データを上記仕様 6 の要求シーケンス仕様に含まれる遷移データの中から抽出すると、得られる差分シーケンス仕様は次の仕様 7 のようになる。

仕様 7：{<表示ボタン入力, お休みタイマ待ち, 時間表示, (電源 OFFまでの時間表示, 表示タイマ起動)>, <表示タイマカウント終了, 時間表示, お休みタイマ待ち, 表示消去>}

【0060】次に、上記仕様 6 の要求シーケンス仕様から上記仕様 5 の部品シーケンス仕様および上記仕様 7 の差分シーケンス仕様と重複している部分を取り除いて次の仕様 8 のような接合シーケンス仕様を作成する（ステップ 105）。

仕様 8：{<お休みタイマボタン入力, 選局処理, お休みタイマ待ち, お休みタイマ起動>, <お休みタイマカウン

ト終了、お休みタイマ待ち、電源OFF、電源リレーランス》

【0061】その後、上記仕様7の差分シーケンス仕様に基づいて既存のシーケンス部品とともに用いられる新たなシーケンス部品を作成するとともに（ステップ106）、上記仕様8の接合シーケンス仕様に基づいて既存のシーケンス部品および新たなシーケンス部品間を接続する接合部品を作成する（ステップ107）。

【0062】図16は上記仕様7の差分シーケンス仕様を実現するシーケンス部品TvSequence2をプログラミング言語（C++言語）により実装した場合の一例を示す図である。なお、図16に示すプログラムの構造は図15に示すプログラムの構造とほぼ同一である。

【0063】上記仕様7の差分シーケンス仕様においては、“表示ボタン入力”と“表示タイマカウント終了”という2種類のイベントが存在するので、これらのイベントを受け付けるようイベント入力手段1aを作成する。イベントの入力はプログラム上では関数の呼出しに対応させることができる。図16に示すプログラムにおいては、イベント入力手段1aとして2つの関数displayButtonEvent()、displayTimeOverEvent()を有し、それぞれ外部からの“表示ボタン入力”イベントおよび“表示タイマカウント終了”イベントの入力をこれらの関数の外部からの呼出しとして受け付けることができる。イベントが入力されると、イベントに対応した状態の変更とアクションの起動とが状態変更／アクション起動手段1dにより行われるが、図16に示すプログラムでは関数displayButtonEvent()、displayTimeOverEvent()の関数本体プログラムがこれに対応する。例えば、関数displayButtonEvent()の関数本体プログラムとして実現されている状態変更／アクション起動手段1dでは、if文による条件分岐により、変数stateが“SLEEP_TIMER”であるか“TIME_DISPLAY”であるかに基づいて、現在の状態が“お休みタイマ待ち”状態であるか“時間表示”状態であるかが判定され、“お休みタイマ待ち”状態であったときにのみイベントに対応する状態の変更とアクションの起動とが行われるようになっている。

【0064】また、アクションの動作内容を実現するアクション出力手段1fは、“電源OFFまでの時間表示”、“表示タイマ起動”および“表示消去”的アクションにそれぞれ対応する関数displayTime()、startDisplayTimer()、eraseTime()として実現されている。それぞれの関数の関数本体プログラムはここでは省略しているが、TV制御システムでは実デバイスである制御用ICに対して動作を指令するデータを送信するためのプログラムやそのようなサブルーチンの呼出しといった実際の処理が実装される。シーケンス部品の現在の状態を保持する現在状態ストア手段1eは変数stateとして実現されている。変数stateは列挙型Stateの変数であり、値として“SLEEP_TIMER”または“TIME_DISPLAY”をとる

が、それぞれの値をとったときにシーケンス部品の現在の状態が“お休みタイマ待ち”状態または“時間表示”状態となる。外部から状態を強制的に変更する状態指定手段1bと、外部から状態について問い合わせを行う状態問い合わせ手段1cはそれぞれ関数setState(State tmpState)、isState(State tmpState)として実現されている。関数setState(State tmpState)は引数として与えられた値を関数本体プログラム中で変数stateに代入することにより指定状態への変更を実現している。また、

10 関数isState(State tmpState)は引数として与えられた値を関数本体プログラム中で現在の状態の値を保持する変数stateと比較し、その結果を真理値として返すことにより、外部の関数の呼出し側に現在の状態についての情報を受け渡すようになっている。

【0065】なお、部品の状態を初期化する初期化手段1gは初期化関数TvSequence2()として実現されており、ここでは関数本体プログラム中で変数stateに“SLEEP_TIMER”的値を代入することにより、状態を“お休みタイマ待ち”状態に初期化している。

20 【0066】図17は上記仕様8の接合シーケンス仕様を実現する接合部品TvSwitcherをプログラミング言語（C++言語）により実装した場合の一例を示す図である。図17に示すプログラムにおいては、アクティブ部品データストア手段2cは列挙型ActiveSequenceの変数activeSequenceとして実現されている。変数activeSequenceは値として“SEQ1”または“SEQ2”をとり、それぞれの値は現在アクティブとなっているシーケンス部品TvSequence1またはシーケンス部品TvSequence2に対応している。部品通信手段2dはシーケンス部品へのポインタ変数であるseq1Ptr、seq2Ptrにより実現される。これらにはそれぞれシーケンス部品TvSequence1のインスタンスのアドレスと、シーケンス部品TvSequence2のインスタンスのアドレスが保持される。C++言語ではポインタ変数を介してインスタンスの関数を呼び出すことができる。このため、接合部品TvSwitcherはそれのインスタンスの関数として実現されたシーケンス部品のイベント入力手段1a、状態指定手段1bおよび状態問い合わせ手段1cにこれらのポインタ変数を用いてアクセスすることができる。

30 【0067】アクション出力手段2eは上記仕様8の接合シーケンス仕様に含まれるアクションの実行の処理を記述するが、図17に示す接合部品TvSwitcherでは“電源リレオフ”および“お休みタイマ起動”的アクションのそれぞれに対応する関数powerOff()、startSleepTimer()として実現されている。それぞれの関数の関数本体プログラムはここでは省略しているが、TV制御システムでは実デバイスである制御用ICに対して動作を指令するデータを送信するためのプログラムやそのようなサブルーチンの呼出しといった処理が実装される。

40 【0068】初期化手段2fは初期化関数TvSwitcher(T

vSequence1* apSeq1, TvSequence2*apSeq2)として実現されている。この初期化関数は引数としてシーケンス部品TvSequence1のインスタンスのアドレスと、シーケンス部品TvSequence2のインスタンスのアドレスとを受け取り、これを部品通信手段2 d であるポインタ変数seq1Ptr, seq2Ptrに代入する。また、この初期化関数はアクティブ部品データストア手段2 c である変数activeSequenceに“SEQ1”を代入することにより、アクティブとなるシーケンス部品をシーケンス部品TvSequence1に初期化している。

【0069】ここで、接合部品TvSwitcherは要求シーケンス仕様に含まれる全てのイベントの入力が可能である必要がある。従って、上記仕様6に含まれる“電源ボタン入力”、“選局ボタン入力”、“お休みタイマカウント終了”、“お休みボタン入力”、“表示ボタン入力”、および“表示タイマカウント終了”の6つのイベントについて入力が可能である必要があり、このため上述したシーケンス部品TvSequence1, TvSequence2と同様にそれぞれのイベントの入力に対応する関数をイベント入力手段2 a として実現する。これらの関数はpowerButtonEvent(), selectButtonEvent(char ch), displayButtonEvent(), displayTimeOverEvent(), sleepButtonEvent(), sleepTimeOverEvent()である。

【0070】イベントが入力されると、これに対応する処理がイベント対応処理手段2 b により行われるが、図17に示すプログラムではイベント入力手段2 a である関数の関数本体プログラムがこれに対応する。イベント対応処理手段2 b では受け付けたイベントと現在の状態とに基づいて部品切替え条件の判定処理、部品切替え条件が成立した場合の処理と不成立な場合の処理を行う。

【0071】まず、部品切替え条件の判定処理については、入力されたイベントをe、アクティブなシーケンス部品の現在の状態をSとするとき、eをイベント（第1要素）、Sを遷移元の状態（第2要素）とする遷移データが接合シーケンス仕様に存在するか否かを判定する。ここでは、接合シーケンス仕様は2つの遷移データから構成されているので、接合シーケンス仕様中の遷移データにおけるイベントと遷移元の状態との組みは <お休みタイマボタン入力、選局処理> と <お休みタイマカウント終了、お休みタイマ待ち> に限定できる。

【0072】従って、部品切替え条件の判定処理は、入力されたイベントが“お休みタイマカウント終了”または“お休みボタン入力”であるときのみで十分であり、それ以外のイベントが入力されたときには部品切替え条件は不成立である。言い換えれば、関数sleepButtonEvent(), sleepTimeOverEvent()以外の関数の呼出しによるイベント入力は関数が呼び出された時点で部品切替え条件が不成立であることが確定しているので、部品切替え条件の判定処理を省略することができる。従って、関数

10 sleepButtonEvent(), sleepTimeOverEvent()の関数本体プログラム中にのみこの判定処理を組み込んでいる。

【0073】例えば、関数sleepButtonEvent()の関数本体プログラムでは2重に入れ子になったif文の外側の条件判定でアクティブ部品データストア手段2 c である変数activeSequenceを参照して“選局処理”状態を持つ方のシーケンス部品がアクティブであるか否かを判定し、さらに内側のif文でこのシーケンス部品の状態問い合わせ手段1 c であるisState()関数を呼び出すことにより、現在の状態が“選局処理”状態であるか否かを判定している。これにより、関数が呼ばれた時点で“お休みタイマボタン入力”イベントの入力は判明していることと合わせ、入力されたイベントが“お休みタイマボタン入力”であり、かつアクティブな部品の現在の状態が“選局処理”であるという部品切替え条件の判定を実現している。

【0074】部品切替え条件が成立した場合、入力されたイベントに対応するアクションの起動、アクティブなシーケンス部品の切替え、新しくアクティブとなったシーケンス部品の内部状態の変更という3つの処理が必要になる。関数sleepButtonEvent()の関数本体プログラムではイベントの入力に対応するアクションの起動としてアクション出力手段2 e の関数startSleepTimer()を呼び出し、アクティブなシーケンス部品の切り替えとしてアクティブ部品データストア手段2 c に新しくアクティブとなるシーケンス部品を表すデータを代入し、新しくアクティブとなったシーケンス部品の内部状態の変更のために部品通信手段2 d であるポインタ変数seq1Ptrを用いてsetState()関数を呼び出すことにより、アクティブなシーケンス部品の状態を“お休みタイマ待ち”状態に設定している。

【0075】部品切替え条件が不成立な場合、接合部品TvSwitcherが受け取ったイベントを現在アクティブなシーケンス部品に対して入力する処理を行う。例えば、関数powerButtonEvent()では、この関数が呼び出された時点で部品切替え条件は不成立であることが確定しているので、関数本体プログラムでは部品切替え条件が不成立な場合の処理のみを実装している。ただし、“電源ボタン入力”イベントを受け付けるのは上記仕様5の部品シーケンス仕様を実現するシーケンス部品のみであるので、このシーケンス部品がアクティブな場合にのみこのシーケンス部品に対してイベントを送信するようになっている。

【0076】次に、このようにして生成されたTV制御システムの作用について説明する。

【0077】図15および図16に示す各シーケンス部品、および図17に示す接合部品は、C++言語のクラスとして実現されており、これらを利用する際にはこれらのクラスのインスタンスを生成する。この際、初期化関数として実現された各部品の初期化手段1 g, 2 f が

動作する。これにより、図15に示すシーケンス部品TvSequence1では“電源OFF”状態に、図16に示すシーケンス部品TvSequence2では“お休みタイマ待ち”状態に初期化される。また、接合部品TvSwitcherでは、部品通信手段2dであるポインタ変数に各シーケンス部品TvSequence1, TvSequence2のインスタンスのアドレスを代入することにより、接合部品と各シーケンス部品との通信路を確保するとともに、アクティブ部品データストア手段2cである変数activeSequenceにシーケンス部品TvSequence1がアクティブであることを示す値“SEQ1”を代入する。

【0078】以上のような初期化処理が終了すると、TV制御システムはイベントを受け付けることが可能となる。外部からのイベントはイベント入力手段2aである各関数を呼び出すことにより行われる。以下、イベントが呼び出されてからの処理の流れをいくつかの場合を想定して説明する。

【0079】まず、第1に、シーケンス部品TvSequence1がアクティブでかつ“電源OFF”状態のときに“電源ボタン入力”イベントを入力した場合の処理の流れについて説明する。

【0080】“電源ボタン入力”イベントの入力は接合部品TvSwitcherの関数powerButtonEvent()を呼び出すことにより行われる。関数powerButtonEvent()は“電源ボタン入力”イベントを受け付けるための接合部品TvSwitcherのイベント入力手段2aである。関数powerButtonEvent()が呼び出されると、イベント対応処理手段2bにおいて“電源ボタン入力”イベントに対応した動作が判定されて実行される。この場合、上述したように、イベントが入力された時点で部品切替え条件は不成立であるので、このイベントのアクティブなシーケンス部品への転送のみが行われる。“電源ボタン入力”イベントを受け付けるのはシーケンス部品TvSequence1のみであるので、イベント対応処理はシーケンス部品TvSequence1が現在アクティブであるか否かを判定する。シーケンス部品TvSequence1がアクティブである場合には、シーケンス部品TvSequence1の“電源ボタン入力”イベントを受け付けるイベント入力手段1aである関数powerButtonEvent()を呼び出す。シーケンス部品TvSequence1は、その関数powerButtonEvent()が呼び出されると、イベント入力手段1aにおいて現在状態ストア手段1eである変数stateを参照し、現在の状態が“電源OFF”状態であるか否かを判定する。“電源OFF”状態である場合には、“電源リレオング”アクションを実行するアクション出力手段1fである関数powerOn()を呼び出す。なお、以上の処理の流れを図19のイベントトレース図に示す。

【0081】次に、第2に、シーケンス部品TvSequence1がアクティブでかつ“選局処理”状態のときに“お休みタイマボタン入力”イベントを入力した場合の処理の

流れについて説明する。

【0082】“お休みタイマボタン入力”イベントの入力は接合部品TvSwitcherの関数sleepButtonEvent()を呼び出すことにより行われる。関数sleepButtonEvent()は“お休みタイマボタン入力”イベントを受け付けるための接合部品TvSwitcherのイベント入力手段2aである。関数sleepButtonEvent()が呼び出されると、イベント対応処理手段2bにおいて“お休みタイマボタン入力”イベントに対応した動作が判定されて実行される。この場合、まず、部品切替え条件が判定されるが、“お休みタイマボタン入力”イベントが入力されたときの部品切替え条件は「TvSequence1がアクティブであり、かつ“選局処理”状態である」ことである。この部品切替え条件を判定するため、接合部品TvSwitcherはまず、アクティブ部品データストア手段2cである変数activeSequenceを参照し、この値が“SEQ1”であるか否かを判定することにより、シーケンス部品TvSequence1がアクティブであるか否かを判断する。さらに、シーケンス部品TvSequence1がアクティブである場合には、このシーケンス部品TvSequence1が“選局処理”状態であることを判定するため、シーケンス部品TvSequence1との通信を行うための部品通信手段2dであるポインタ変数seq1Ptrを介してシーケンス部品TvSequence1の状態問い合わせ手段1bである関数isState()に対して、“選局処理”状態を示す“TvSequence1::TUNING”という値を引数に与えて呼び出す。シーケンス部品TvSequence1はこの関数isState()の呼出しに対して、引数として渡された値が現在状態ストア手段1eの現在の値と等しいか否かの真理値を関数の戻り値として返す。接合部品TvSwitcherはこれを受けて、シーケンス部品TvSequence1の状態が“選局処理”である場合、部品切替え条件が成立したと判定し、“お休みタイマボタン入力”イベントに対応するアクションである“お休みタイマ起動”アクションの実行、アクティブなシーケンス部品となるTvSequence2への切替え、シーケンス部品TvSequence2の状態の“お休みタイマ待ち”状態への強制的な変更を行う。“お休みタイマ起動”アクションの実行はこれに対応するアクション出力手段2eである関数startSleepTimer()の呼出しにより行われる。また、アクティブなシーケンス部品の切替えはアクティブ部品データストア手段2cである変数activeSequenceの値をTvSequence2がアクティブであることを表す値である“SEQ2”に変更することにより行われる。また、状態の強制的な変更はシーケンス部品TvSequence1, TvSequence2との通信を行うための部品通信手段2dであるポインタ変数seq2Ptrを介してシーケンス部品TvSequence2の状態指定手段1bである関数setState()に対して、“お休みタイマ待ち”状態を表す値“TvSequence2::SLEEP_TIMER”を引数に与えて呼び出すことにより行われる。シーケンス部品TvSequence2では関数setState()が呼び出されると、現在状態ストア手段

1 e である変数stateに引数で渡された値 “TvSequence2::SLEEP_TIMER” を代入することにより、状態を “お休みタイマ待ち” 状態に変更する。なお、以上の処理の流れを図19のイベントトレース図に示す。

【0083】このように第1の実施例によれば、シーケンス部品TvSequence1が既に存在する場合に、シーケンス部品TvSequence1の内部を変更することなく、追加的な部品TvSequence2, TvSwitcherを新たに作成することにより、図14に示すような拡張した動作仕様を実現するTV制御システムを生成することができる。これにより、TV制御システムの開発者は既存のシーケンス部品TvSequence1の内部について詳細に理解する必要はなく、図13に示すような状態遷移図により記述される動作仕様のみを把握することにより開発を進めることができるので、新たなTV制御システムを効率的に開発することが可能となる。

【0084】第2の実施例

次に、図20乃至図22により、図1乃至図12に示すシーケンス制御システムの別の実施例について説明する。なお、ここでは、図10乃至図12に示すシーケンス部品および接合部品を、オブジェクト指向言語であるC++言語でなく、関数型言語であるC言語で実装する場合について説明する。

【0085】図20に示すプログラムは、上記仕様1の部品シーケンス仕様（図4参照）を実現するシーケンス部品である。図20において、イベント入力手段1aは関数Component_E1(), Component_E2()に対応している。外部からのイベントの入力はこれらの関数を呼び出すことにより行われる。アクション出力手段1fは関数Component_A1(), Component_A2()に対応している。これらの関数の内部では、例えば出力バッファへのデータの出力やタイマの起動等の様々なアクションが実装されるが、ここでは詳細な説明は省略する。現在状態ストア1eはグローバル変数Component_stateにより実現される。グローバル変数Component_stateは値として “Alpha” または “Beta” をとり、それぞれの値をとったときにシーケンス部品Componentの現在の状態が “ α ” または “ β ” となる（図4参照）。初期化手段1gは初期化関数initComponent()として実現される。この初期化関数はシーケンス部品を利用するときに外部から呼び出されることにより、グローバル変数Component_stateの値を “Alpha” にし、このシーケンス部品の初期状態を “ α ” に設定する。状態変更/アクション出力手段1dは関数Component_E1(), Component_E2()の関数本体プログラムにより実現されている。関数Component_E1(), Component_E2()は関数呼出しが発生する度に、現在のstateの値に基づいて、次のstateへの変化を求め、状態遷移が必要な場合にはその値を書き換える。状態指定手段1bは関数Component_setState()により実現される。この関数は引数として与えられた値をstateを代入することにより指

定状態への変化を実現している。また、状態問い合わせ手段1cは関数Component_isState()により実現される。この関数は現在の状態が引数として与えられた状態であるか否かを判定し、その結果を返す。

【0086】図21に示すプログラムは、上記仕様3の差分シーケンス仕様（図7参照）を実現するシーケンス部品である。図21に示すように、このプログラムの構造は図20に示すプログラムの構造と略同一であるので、詳細な説明は省略する。

10 【0087】図22に示すプログラムは、上記仕様4の接合シーケンス仕様（図8参照）を実現する接合部品である。図22において、イベント入力手段2aは関数Switcher_E1()～Switcher_E4()に対応している。外部からのイベントの入力はこれらの関数を呼び出すことにより行われる。アクティブ部品データストア2cは変数Switcher_activeSequenceにより実現される。この変数は値として “COMPONENT” または “COMPLEMENT” をとり、それぞれの値をとったときにシーケンス部品ComponentまたはComplementのいずれかがアクティブであることを表

20 している。部品通信手段2dはここでは各シーケンス部品のイベント入力手段1a、状態指定手段1b、および状態問い合わせ手段1cである各関数が兼ねている。すなわち、各関数はC言語の外部関数として定義されているので、例えばシーケンス部品Componentの関数Component_isState()を接合部品Switcherの関数Switcher_E3()から直接呼び出すことができる。

【0088】初期化手段2fは初期化関数initSwitcher()により実現される。初期化関数initSwitcher()はアクティブ部品データストア手段2cである変数Switcher_activeSequenceの値を “COMPONENT” に初期化する。これにより、アクティブなシーケンス部品をComponentに初期化する。

30 【0089】イベント対応処理手段2bは関数Switcher_E3(), Switcher_E4()の関数本体プログラムにより実現されている。部品切替え条件の判定は、Switcher_E3(), Switcher_E4()の内部の2つのif文で実現されている。部品切替え条件の成立時のアクションの実行の処理、切替え先となるシーケンス部品の内部状態を変更する処理、および切替え先となるシーケンス部品をアクティブとして他をインアクティブとする処理は、関数Switcher_E3()の本体プログラムでは関数Switcher_A3()の呼び出し、関数Component_setState()の呼び出し、変数Switcher_activeSequenceの “COMPONENT” への値の書き換えにより実現される。また、関数Switcher_E4()の関数本体プログラムでは関数Switcher_A4()の呼び出し、関数Complement_setState()の呼び出し、変数Switcher_activeSequenceの “COMPLEMENT” への値の書き換えにより実現される。一方、部品切替え条件の不成立時の処理は、関数Switcher_E1(), Switcher_E2()の内部におけるアクティブなシーケンス部品へのイベント受け付け関数Component_

E1(), Complement_E1()の呼出しにより実現される。なお、アクション出力手段2 eには関数Switcher_A3(), Switcher_A4()が対応している。

【0090】このように、第2の実施例によれば、関数型言語であるC言語によっても、第1の実施例と同様に新たなシーケンス制御システムを生成することができ、サブシステム化（部品化）によりシステム開発を効率的に進めることができる。

【0091】

【発明の効果】以上説明したように本発明によれば、状態遷移動作システムを再利用して新たな状態遷移動作システムを生成することができるので、新たな状態遷移動作システムを効率的に開発することができる。また、このような開発過程で作成された状態遷移部品や接合部品についても次の新たな開発過程でシーケンス部品として再利用することができるので、状態遷移動作システムをサブシステム化（部品化）してシステム開発を効率的に進めることができる。

【図面の簡単な説明】

【図1】本発明による状態遷移動作システム（シーケンス制御システム）の一実施の形態を示すブロック図。

【図2】図1に示すシーケンス部品の詳細を示すブロック図。

【図3】図1に示す接合部品の詳細を示すブロック図。

【図4】再利用される既存のシーケンス制御システム（シーケンス部品）の動作仕様を記述した状態遷移図。

【図5】要求されるシーケンス制御システムの動作仕様を記述した状態遷移図。

【図6】図1に示すシーケンス制御システムの生成方法を説明するためのフローチャート。

【図7】図4および図5の状態遷移図に基づいて作成された差分シーケンス仕様に対応する状態遷移図。

【図8】図4および図5の状態遷移図に基づいて作成された接合シーケンス仕様に対応する状態遷移図。

【図9】図1および図3に示す接合部品における処理内容を説明するためのフローチャート。

【図10】図4に示す動作仕様を実現するシーケンス部品を実装するためのプログラムの一例を示す図。

【図11】図7に示す動作仕様を実現するシーケンス部品を実装するためのプログラムの一例を示す図。

【図12】図8に示す動作仕様を実現する接合部品を実装するためのプログラムの一例を示す図。

【図13】再利用される既存のTV制御システム（シーケンス部品）の動作仕様を記述した状態遷移図。

【図14】要求されるTV制御システムの動作仕様を記

述した状態遷移図。

【図15】図13に示す動作仕様を実現するシーケンス部品を実装するためのプログラムの一例を示す図。

【図16】差分シーケンス仕様を実現するシーケンス部品を実装するためのプログラムの一例を示す図。

【図17】接合シーケンス仕様を実現するシーケンス部品を実装するためのプログラムの一例を示す図。

【図18】TV制御システムにおける処理の流れの一例を説明するためのイベントトレース図。

10 【図19】TV制御システムにおける処理の流れの別の例を説明するためのイベントトレース図。

【図20】図4に示す動作仕様を実現するシーケンス部品を実装するためのプログラムの別の例を示す図。

【図21】図7に示す動作仕様を実現するシーケンス部品を実装するためのプログラムの別の例を示す図。

【図22】図8に示す動作仕様を実現する接合部品を実装するためのプログラムの別の例を示す図。

【図23】シーケンス制御システム（TV制御システム）の動作仕様の一例を記述した状態遷移図。

20 【図24】図23に示すシーケンス制御システムをプログラミング言語（C++言語）により実装した場合のシーケンス制御プログラムの一例を示す図。

【図25】図23に示すシーケンス制御システム（TV制御システム）に“お休みタイム”的機能を追加した場合の動作仕様の一例を記述した状態遷移図。

【図26】図24に示すシーケンス制御プログラムの内部を変更して作成した、図25に示す動作仕様を実現するシーケンス制御プログラムの一例を示す図。

【符号の説明】

30 1 シーケンス部品（状態遷移部品）

1 a イベント入力手段

1 b 状態指定手段

1 c 状態問い合わせ手段

1 d 状態変更／アクション起動手段

1 e 現在状態ストア手段

1 f アクション出力手段

1 g 初期化手段

2 接合部品

2 a イベント入力手段

40 2 b イベント対応処理手段

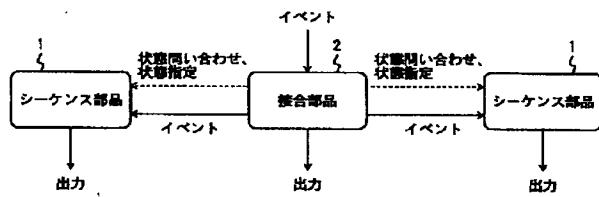
2 c アクティブ部品データストア手段

2 d 部品通信手段

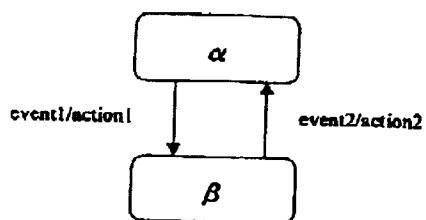
2 e アクション出力手段

2 f 初期化手段

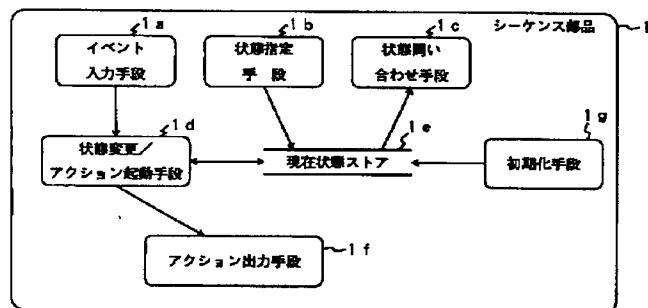
【図1】



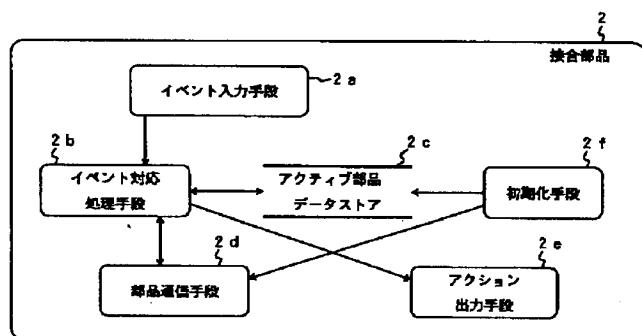
【図4】



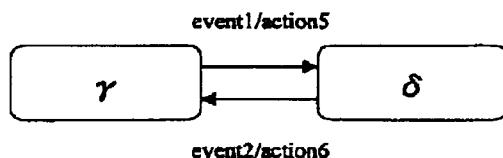
【図2】



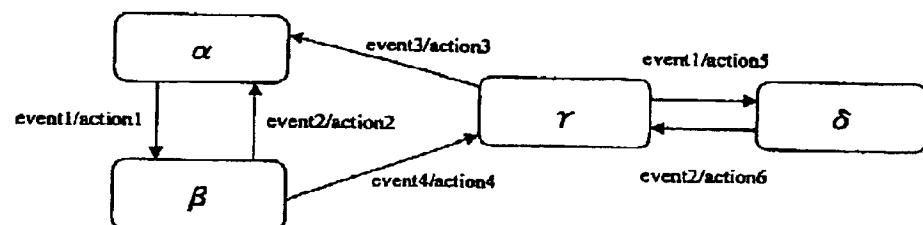
【図3】



【図7】



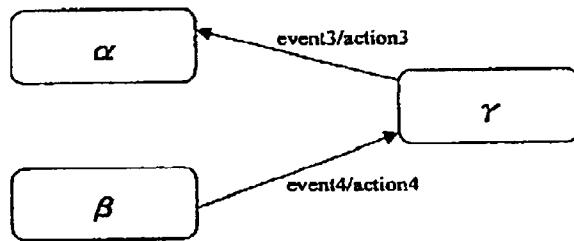
【図5】



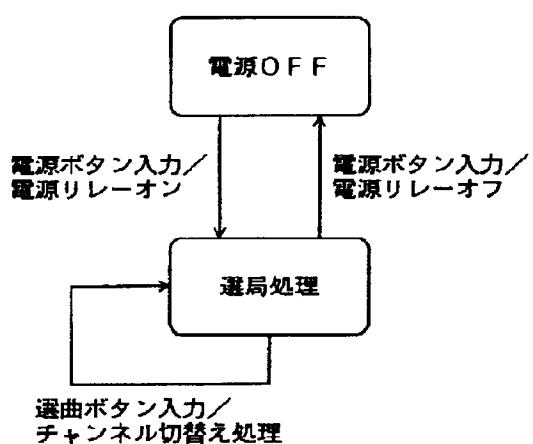
【図6】



【図8】

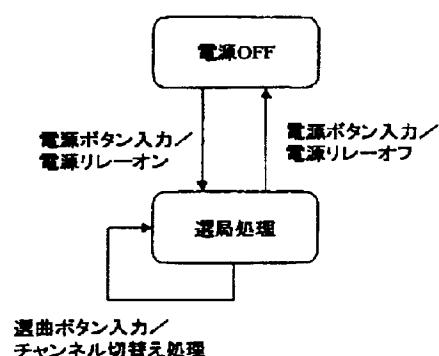
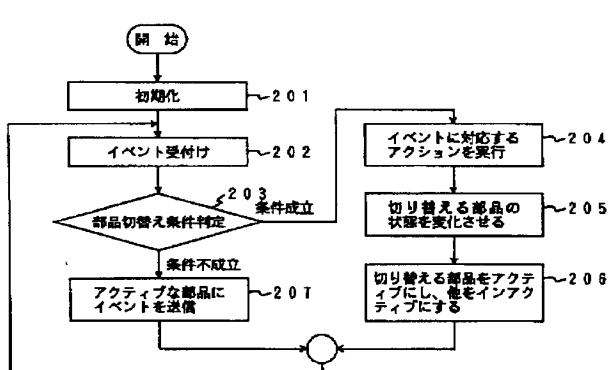


【図13】



【図23】

【図9】



【図10】

```
class Component {
public:
    enum State {Alpha, Beta};
private:
    State state;                                // 現在状態ストア手段 |e

    void action10{/* action1 の実装 */};          // アクション
    void action20{/* action2 の実装 */};          // 出力手段 |f

public:
    Component(){ state = Alpha; }                  // 初期化手段 |g

    void setState(State tmpState){ state = tmpState; } // 状態指定手段 |b
    int isState(State tmpState){ return int(state == tmpState); } // 状態問い合わせ手段 |c

    void event10{
        if(state == Alpha){ action10; state = Beta; } // イベント入力手段 |a
    }
    void event20{
        if(state == Beta){ action20; state = Alpha; }
    }
};
```

状態変更／アクション起動手段 |d

【図11】

```
class Complement {
public:
    enum State { Gamma, Delta };
private:
    State state;                                // 現在状態ストア手段 |e

    void action50{/* action5 の実装 */};          // アクション
    void action60{/* action6 の実装 */};          // 出力手段 |f

public:
    Complement(){ state = Gamma; }                  // 初期化手段 |g

    void setState(State tmpState){ state = tmpState; } // 状態指定手段 |b
    int isState(State tmpState){ return int(state == tmpState); } // 状態問い合わせ手段 |c

    void event10{
        if(state == Gamma){ action50; state = Delta; } // イベント入力手段 |a
    }
    void event20{
        if(state == Delta){ action60; state = Gamma; }
    }
};
```

状態変更／アクション起動手段

【図12】

```

class Switcher {
    Component *component;
    Complement *complement;
    --- 部品通信手段 2d

    int ComponentActiveFlag;
    int ComplementActiveFlag;
    --- アクティブ部品
    --- データストア手段 2c

    void action30{/* action3 の実装 */};
    void action40{/* action4 の実装 */};
    --- アクション出力手段 2e

public:
    Switcher(Component *pCompo, Complement *pCompl){
        component = pCompo;
        complement = pCompl;
        ComponentActiveFlag = 1;
        ComplementActiveFlag = 0;
    }
    --- 初期化手段 2f

    void event10{
        if (ComponentActiveFlag) component->event10();
        if (ComplementActiveFlag) complement->event10();
    }
    void event20{
        if (ComponentActiveFlag) component->event20();
        if (ComplementActiveFlag) complement->event20();
    }
    void event30{
        if (ComplementActiveFlag){
            if (complement->isState(Complement::Gamma)){
                action30();
                component->setState(Component::Alpha);

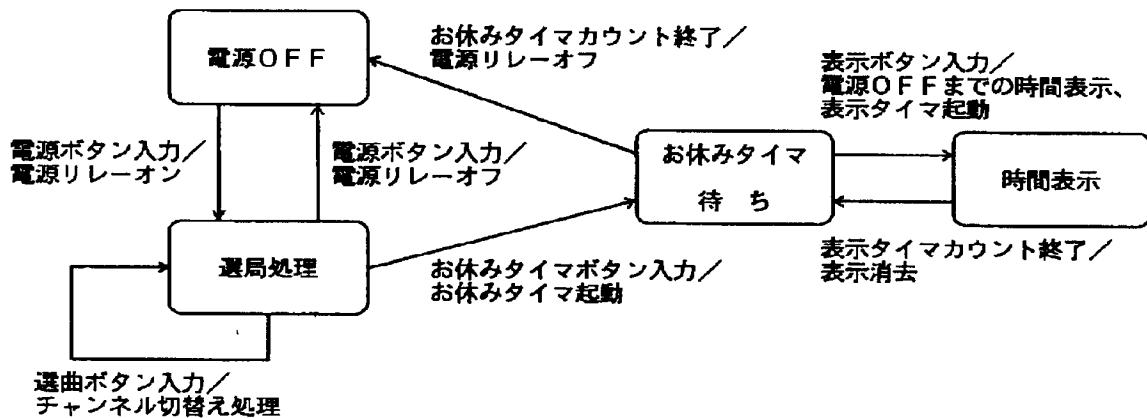
                ComponentActiveFlag = 1;
                ComplementActiveFlag = 0;
            }
        }
    }
    void event40{
        if (ComponentActiveFlag){
            if (component->isState(Component::Beta)){
                action40();
                complement->setState(
                    Complement::Gamma);

                ComponentActiveFlag = 0;
                ComplementActiveFlag = 1;
            }
        }
    }
};
    --- イベント
    --- 入力手段
    --- 2a

    --- イベント
    --- 対応処理
    --- 手段
    --- 2b

```

【図14】



【図15】

```

class TvSequence1 {
public:
    enum State { POWER_OFF, TUNING };
private:
    State state; // 現在状態ストア手段 |e|
    void powerOn0{/*リレーオンアクションの実装*/};
    void powerOff0{/*リレーオフアクションの実装*/};
    void setChannel(char ch){/*チャンネル切替え処理の実装*/};
public:
    TvSequence10{state = POWER_OFF}; // 初期化手段 |g|
    void setState(State tmpState){state = tmpState}; // 状態指定手段 |b|
    int isInState(State tmpState){return int(state == tmpState);}; // 状態問い合わせ手段 |c|
    void powerButtonEvent(){
        if (state == POWER_OFF){
            powerOn0();
            state = TUNING;
        }
        else if (state == TUNING){
            powerOff0();
            state = POWER_OFF;
        }
    }
    void selectButtonEvent(char ch){
        if (state == TUNING){
            setChannel(ch);
        }
    }
};
  
```

Diagram annotations for Figure 15:

- 現在状態ストア手段 |e|: Points to the `state` variable in the private section.
- アクション出力手段 |f|: Points to the `powerOn0()`, `powerOff0()`, and `setChannel()` methods.
- 初期化手段 |g|: Points to the constructor `TvSequence10{state = POWER_OFF};`.
- 状態指定手段 |b|: Points to the `setState()` method.
- 状態問い合わせ手段 |c|: Points to the `isInState()` method.
- 状態変更/アクション起動手段 |d|: Points to the `powerButtonEvent()` method.
- イベント入力手段 |a|: Points to the `selectButtonEvent()` method.

【図16】

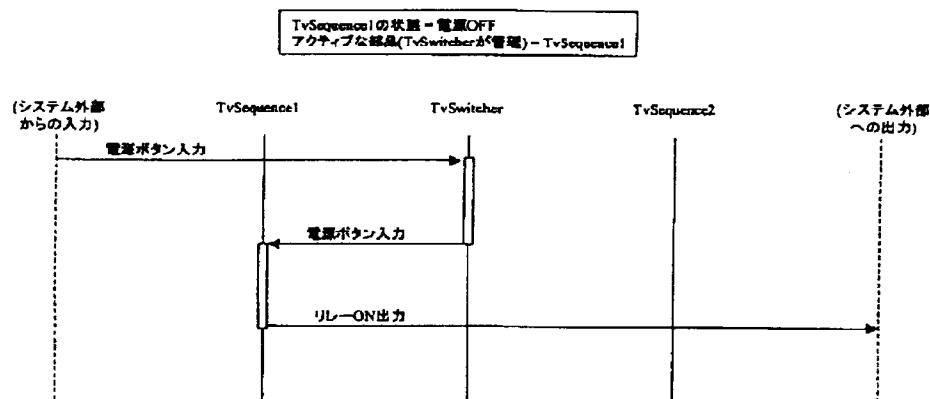
```

class TvSequence2 {
public:
    enum State { SLEEP_TIMER, TIME_DISPLAY };
private:
    State state;
    void displayTime0{/*電源 OFFまでの時間表示アクションの実装*/}
    void eraseTime0{/*表示消去アクションの実装*/}
    void startDisplayTimer0{/*表示タイマ起動アクションの実装*/}
public:
    TvSequence20( state = SLEEP_TIMER );
    void setState(State tmpState){ state = tmpState; }
    int isState(State tmpState){ return int(state == tmpState); }
    void displayButtonEvent0{
        if (state == SLEEP_TIMER) {
            displayTime0;
            startDisplayTimer0;
            state = TIME_DISPLAY;
        }
    }
    void displayTimeOverEvent0{
        if (state == TIME_DISPLAY) {
            eraseTime0;
            state = SLEEP_TIMER;
        }
    }
};

```

現在状態ストア手段 |e
アクション出力手段 |f
初期化手段 |g
状態指定手段 |b
状態問い合わせ手段 |c
状態変更／アクション起動手段 |d
イベント入力手段 |a

【図18】



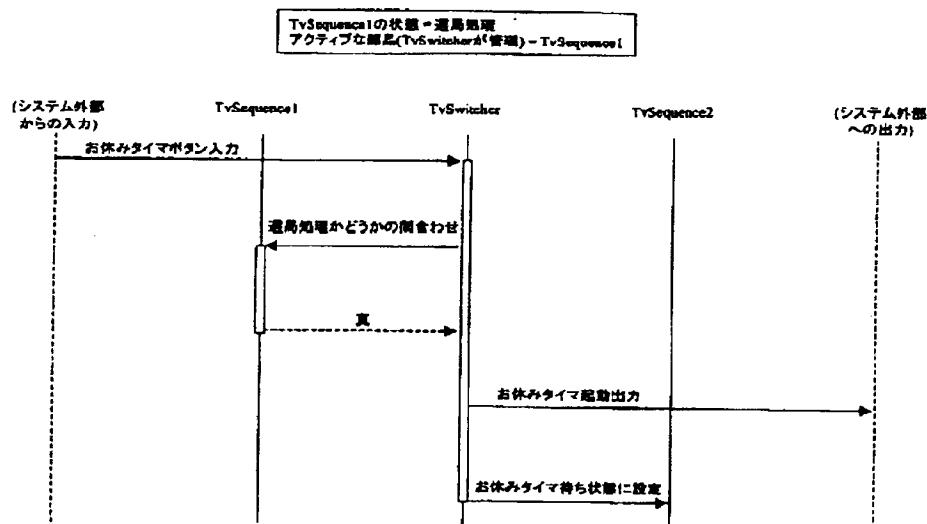
【図17】

```

class TvSwitcher {
    enum ActiveSequence { SEQ1, SEQ2 };
    ActiveSequence activeSequence;
    TvSequence1* seq1Ptr;
    TvSequence2* seq2Ptr;
    void powerOff() /*リレーオフアクションの実装*/;
    void startSleepTimer() /*お休みタイマ起動アクションの実装*/;
public:
    TvSwitcher(TvSequence1* apSeq1, TvSequence2* apSeq2){
        seq1Ptr = apSeq1; seq2Ptr = apSeq2;
        activeSequence = SEQ1;
    }
    void powerButtonEvent(){
        if(activeSequence == SEQ1) seq1Ptr->powerButtonEvent();
    }
    void selectButtonEvent(char ch){
        if(activeSequence == SEQ1) seq1Ptr->selectButtonEvent(ch);
    }
    void displayButtonEvent(){
        if(activeSequence == SEQ2) seq2Ptr->displayButtonEvent();
    }
    void displayTimeOverEvent(){
        if(activeSequence == SEQ2) seq2Ptr->displayTimeOverEvent();
    }
    void sleepButtonEvent(){
        if(activeSequence == SEQ1) {
            if(seq1Ptr->isState(TvSequence1::TUNING)){
                startSleepTimer();
                activeSequence = SEQ2;
                seq2Ptr->setState(TvSequence2::SLEEP_TIMER);
            }
        }
    }
    void sleepTimeOverEvent(){
        if(activeSequence == SEQ2) {
            if(seq2Ptr->isState(TvSequence2::SLEEP_TIMER)){
                powerOff();
                activeSequence = SEQ1;
                seq1Ptr->setState(TvSequence1::POWER_OFF);
            }
        }
    }
};

```

【図19】



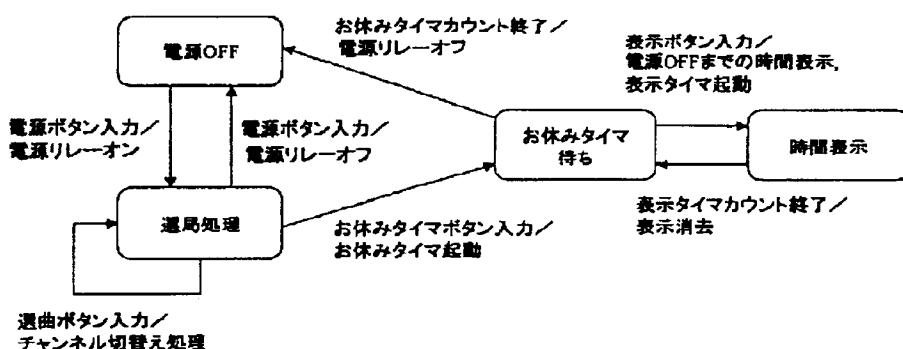
【図20】

```

typedef enum {Alpha, Beta} Component_State;
Component_State Component_state;           // 現在状態ストア手段1 a
void Component_A1() { / / アクションA.1の実現 / / }           // アクション出力手段1 f
void Component_A2() { / / アクションA.2の実現 / / }           // アクション出力手段1 f
void InitComponent () {Component_state=Alpha; }           // 初期化手段1 g
void Component_setState(Component_State tmpState) {Component_state=tmpState;}           // 状態指定手段1 b
int Component_isState (Component_State
    return int (Component_state==tmpState);           // 状態問合せ手段1 c
}

void Component_E1() {
    if (Component_state==Alpha) {
        Component_A1();
        Component_state= Beta;
    }
}
void Component_E2() {
    if (Component_state==Beta) {
        Component_A2();
        Component_state=Alpha;
    }
}
  
```

【図25】



【図21】

```

typedef enum ( Gamma, Delta ) Complement_State;
Complement_State Complement_state; ~現在状態ストア手段1 e

void Complement_A5() /*アクションA5の実装*/; ~アクション出力手段1 f
void Complement_A6() /*アクションA6の実装*/; ~初期化手段1 g
void initComplement () (Complement_state=Gamma); ~状態指定手段1 b

void Complement_setState(Complement_State tmpState) {Complement_state=tmpState;}; ~状態割り当て手段1 c
int Complement_isState (Complement_State tmpState) { return int(Complement_state == tmp_State); }

void Complement_E1() {
    if(Complement_state==Gamma) {
        Complement_A5();
        Complement_state=Delta;
    }
}
void Complement_E2() {
    if(Complement_state==Delta) {
        Complement_A6();
        Complement_state=Gamma;
    }
}

```

【図24】

```

1: class TvSequence {
2: public:
3:     enum State { POWER_OFF, TUNING };
4: private:
5:     State state;
6:
7:     void powerOn() /* "電源リーアン" アクションの実装 */;
8:     void powerOff() /* "電源リーオフ" アクションの実装 */;
9:     void setChannel(char ch) /* "チャンネル切替処理" アクションの実装 */;
10:
11: public:
12:     TvSequence(){ state = POWER_OFF; }
13:
14:     void powerButtonEvent();
15:     if(state == POWER_OFF){
16:         powerOn();
17:         state = TUNING;
18:     }
19:     else if(state == TUNING){
20:         powerOff();
21:         state = POWER_OFF;
22:     }
23: }
24:
25:     void selectButtonEvent(char ch);
26:     if(state == TUNING){
27:         setChannel(ch);
28:     }
29: };
30:

```

【図22】

```

typedef enum { COMPONENT, COMPLEMENT } ActiveSequence;

[ ActiveSequence Switcher_activeSequence; ] --- アクティブ部品
                                            データストア手段 2c

[ void initSwitcher0{ Switcher_activeSequence = COMPONENT; } ] --- 初期化手段 2f

[ void Switcher_A30( /*アクション A3 の実装*/ );
void Switcher_A40( /*アクション A4 の実装*/ ) ] --- アクション出力手段 2e

[ void Switcher_E10{
    if (Switcher_activeSequence == COMPONENT) Component_E10;
    else if (Switcher_activeSequence == COMPLEMENT) Complement_E10;
}
void Switcher_E20{
    if (Switcher_activeSequence == COMPONENT) Component_E20;
    else if (Switcher_activeSequence == COMPLEMENT) Complement_E20;
}

void Switcher_E30{
    if (Switcher_activeSequence == COMPLEMENT){
        if (Complement_isState(Gamma)){
            Switcher_A30;
            Complement_setState(Alpha);
            Switcher_activeSequence = COMPONENT;
        }
    }
}

void Switcher_E40{
    if (Switcher_activeSequence == COMPONENT){
        if (Component_isState(Beta)){
            Switcher_A40;
            Component_setState(Gamma);
            Switcher_activeSequence = COMPLEMENT;
        }
    }
}

```

The diagram illustrates the state transition logic for the Switcher. It includes the following components and annotations:

- Initial State:** The code block `void initSwitcher0{ Switcher_activeSequence = COMPONENT; }` is annotated with **初期化手段 2f**.
- Action Output:** The code blocks `void Switcher_A30()` and `void Switcher_A40()` are annotated with **アクション出力手段 2e**.
- Event Input:** The code blocks `void Switcher_E10()`, `void Switcher_E20()`, `void Switcher_E30()`, and `void Switcher_E40()` are annotated with **イベント入力手段 2a**.
- Data Storage:** The variable `ActiveSequence Switcher_activeSequence;` is annotated with **アクティブ部品 データストア手段 2c**.
- Event Handling:** The code blocks `void Switcher_E30()` and `void Switcher_E40()` are annotated with **イベント対応処理手段 2b**.

【図26】

```

1:  class TvSequence {
2:  public:
3:    enum State { POWER_OFF, TUNING,
4:                  SLEEP_TIMER, TIME_DISPLAY };
5:  private:
6:    State state;
7:
8:    void powerOn() /* "電源リーアン" アクションの実装*/;
9:    void powerOff() /* "電源リーオフ" アクションの実装*/;
10:   void setChannel(char ch) /* "チャンネル切替え処理" アクションの実装*/;
11:   void displayTime() /* "電源 OFF までの時間表示" アクションの実装*/;
12:   void eraseTime() /* "表示消去" アクションの実装*/;
13:
14:   public:
15:     TvSequence() { state = POWER_OFF; }
16:
17:     void powerButtonEvent(){
18:       if(state == POWER_OFF)
19:         powerOn();
20:         state = TUNING;
21:       }
22:       else if (state == TUNING)
23:         powerOff();
24:         state = POWER_OFF;
25:       }
26:
27:
28:     void selectButtonEvent(char ch){
29:       if(state == TUNING)
30:         setChannel(ch);
31:       }
32:     }
33:
34:     void sleepTimerButtonEvent(){
35:       if(state == TUNING) {
36:
37:         //SLEEP_TIME で指定された時間の後に
38:         //メンバ関数 sleepTimeOverEvent() を呼び出す
39:         sleepTimer.start(SLEEP_TIME,
40:                           sleepTimeOverEvent);
41:
42:         state = SLEEP_TIMER;
43:       }
44:
45:
46:     void sleepTimeOverEvent(){
47:       if(state == SLEEP_TIMER) {
48:         powerOn();
49:         state = POWER_OFF;
50:       }
51:
52:
53:     void displayButtonEvent(){
54:       if(state == SLEEP_TIMER) {
55:
56:         //DISPLAY_TIME で指定された時間の後に
57:         //メンバ関数 displayTimeOverEvent() を呼び出す
58:         displayTimer.start(DISPLAY_TIME,
59:                           displayTimeOverEvent);
60:
61:         displayTime();
62:         state = TIME_DISPLAY;
63:       }
64:
65:
66:     void displayTimeOverEvent(){
67:       if(state == TIME_DISPLAY) {
68:         eraseTime();
69:         state = SLEEP_TIMER;
70:       }
71:
72:
73:     }
74:
75:   };

```